САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОЙ ТЕОРИИ ИГР И СТАТИСТИЧЕСКИХ РЕШЕНИЙ

Беляков Иван Владимирович

Выпускная квалификационная работа бакалавра

Программная реализация поиска равновесия по Нэшу на сети MANET

Направление 02.03.02

«Фундаментальная информатика и информационные технологии»

Заведующий кафедрой,
доктор физ.-мат. наук,
профессор
Петросян Л.А.

Научный руководитель,
доктор физ.-мат. наук,
доцент
Громова Е.В.

Рецензент,
ст. преподаватель,
Стученков А.Б.

Санкт-Петербург

2018

Содержание

Введение

В настоящее время использование современных средств связи является приоритетным направлением развития науки и техники. Модернизация устройств с целью увеличения зоны покрытия, качества передаваемой информации и пр. является важной не только для повышения комфортности жизни пользователей данных устройств, но и, в том числе, при спасении жизней людей в районах стихийных бедствий, катастроф, в труднодоступных районах. MANET (Mobile Ad hoc Network) — это беспроводные децентрализованные самоорганизующиеся сети, состоящие из мобильных устройств (узлов). Сети MANET имеют динамическую топологию, т. е. каждый узел может произвольно перемещаться, включаться в сеть и покидать её в любой момент времени. Каждый сетевой узел может связываться только с узлами, размещенными в пределах его радиуса действия (зоны покрытия) [1].

Особенность сетей MANET – это способность реорганизовывать сетевую структуру, используя результаты предыдущих этапов построения. Таким образом можно достичь улучшения общей производительности сети с течением времени [2].

Все узлы имеют некоторую определенную область взаимодействия с другими узлами. Так происходит образование пути для передачи информации. Большое количество узлов между источником и приемником данных отрицательно сказывается на скорости передачи данных [3]. Длительное использование передатчиков негативно влияет на их работу, так как они могут выйти из строя. Уменьшение рабочей нагрузки на сеть или построение маршрута можно совершить, используя подвижное мобильное устройство передачи данных.

Предположим, что имеется несколько команд, участвующих в поисковой операции в труднодоступной местности. Каждый член команды

имеет передатчик, позволяющий ему передавать информацию членам группы. Очевидно, что задача улучшения производительности сети, образованной передатчиками членов команд, является крайне важной.

Данная задача может быть формализована при помощи теоретико-игрового подхода. Имеется несколько игроков (команд) со своим набором агентов (членов команд). Каждый агент находится в некотором узле прямоугольной решетки. Вся эта система образует граф, который и является сетевой структурой.

В данной дипломной работе игроки имеют три типа агентов: так называемые «концы пути», промежуточные вершины и подвижные/мобильные агенты (дроны). Мобильный агент может быть установлен в незанятую вершину решетки, в том числе, в недоступных для обычных агентов районах, и устанавливать связи с неподвижными агентами, а также с подвижными агентами других игроков, улучшая тем самым производительность сети. В контексте данной работы предполагаем, что для каждого из игроков имеется определенный набор подвижных дронов

Путь в графе необходим для оценки времени, необходимого на доставку пакета информации из одной концевой вершины в другую, поэтому затраты игроков будем считать пропорциональными данному значению. Основная решаемая задача в данном исследовании — это поиск подходящего места для перемещения подвижных агентов. Подходящее место — это такая позиция, перемещение подвижного агента в которую улучшает производительность сети либо даёт возможность построить связь между концевыми агентами.

Сетевая структура будет изображаться планарным графом, разложенным в виде прямоугольной решетки. Вершины, расположенные на пересечениях кривых, разбивающих плоскость, - это позиции агентов. Отрезки обозначают связи между агентами в виде ребер графа [4]. Целью

каждого игрока будем считать уменьшение минимального пути между двумя фиксированными вершинами на подграфе, определенного на вершинах игрока. В данном исследовании рассматривается постановка некооперативной многошаговой игры управления агентами сети [5] в условиях нестабильной работы некоторых агентов. Под нестабильностью в работе будет пониматься свойство некоторых агентов выходить из строя, а именно, будем считать заданными вероятности выхода из строя для «промежуточных агентов». В данной задаче предполагаем, что спустя некоторое время, затраченное на восстановление, агент возвращается в рабочее состояние.

В области работы с сетями MANET существуют подобные исследования. Детерминированная задача на прямоугольной решетке была рассмотрена в работе [6]. Детерминированная задача на шестиугольной решётке рассмотрена в работе [7]. Текущее исследование подразумевает работу с прямоугольной решёткой, при этом агенты, участвующие в построении связи, могут выходить из строя. Кроме того, в отличие от работы [6], для каждого игрока решаем задачу минимизации пути между фиксированными узлами.

Задача на сети MANET в кооперативной постановке была рассмотрена в работе [8].

Стоит отметить, что данная игра относится к классу стохастических многошаговых игр [9]. Однако для данного класса игр задача нахождения равновесия по Нэшу является крайне сложной задачей и даже для игры трёх лиц не существует известного метода нахождения равновесия по Нэшу в чистых стратегиях. Ввиду этого, полученное в ходе исследования решение не будет точным, но приближённым. Ситуации, получаемые в рамках исследования, не будут удовлетворять строгому определению равновесия во Нэшу [10], по причине стохастической природы задачи, но будут являться приближёнными к ним [11][12].

Основные цели и задачи

В данной выпускной квалификационной работе должно быть представлено решение некооперативной многошаговой игры. Для этого необходимо решить следующие задачи:

1. На примере трех лиц исследовать данную игру с применением методов Теории Игр;
2. Представить алгоритм решения этой задачи;
3. Реализовать программное обеспечение для исследования данной игры;
4. Проиллюстрировать работу программы на примере игры трех игроков.

Обзор литературы

1. Петросян Л. А, Зенкевич Н. А, Шевкопляс Е. В. Теория игр.

   В этой книге обозначены основные понятия, требующиеся для основы работы с математической теорией игр и исследования операции.

2. Новиков Д. А. Игры и сети. Математическая теория игр и её приложения.

   В данной книге рассматриваются общие положения и способы работы с играми на сетях. Обозначен подход к этим играм с точки зрения теории графов.

3. Ekaterina Gromova, Dmitry Gromov, Nikolay Timonin, Anna Kirpichnikova, Stewart Blakeway. A dynamic game of mobile agents placement in a MANET.

   Исследование основано на задаче, похожей на представленную в статье. Ключевые отличия заключаются в разных подходах к тому, что считать выигрышем, количестве подвижных агентов и в поведениях агентов.

4. Taisiia Plehanova, Ekaterina Gromova, Dmitry Gromov, Stewart Blakeway, Anna Kirpichnikova. The strategic placement of mobile agents on a hexagonal graph using game theory.

   В этой работе проведено похожее исследование, но на шестиугольной решётке, имеющей большее распространение в сетях MANET.

5. Громова Е.В., Плеханова Т.М. Кооперативная динамическая игра на сети MANET, Процессы управления и устойчивость. Труды 49-й международной научной конференции аспирантов и студентов / науч. ред. тома Н. В. Смирнов.

   В данной статье рассматривается практическое применение исследований связанных с сетями MANET.

6. Буре В.М., Парилина Е.М. Стохастические модели передачи данных в сетях с различными топологиями.

   Данная работа даёт обширную теорию про игры, имеющие стохастическую природу.

7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to Algorithms. — 3rd edition.

   Данная книга даёт обширное представление об различных алгоритмах и возможности их применения. В данной книге можно найти описание алгоритмов, позволяющих построить кратчайший путь между вершинами графов.

Глава 1. Постановка задачи

1.1. Структура игры

Игрой в нормальной форме будем называть систему:

$$\Gamma = (N, \{U_i\}_{i \in N}, \{C_i\}_{i \in N}),$$

где N — это множество игроков, U_i — множество стратегий для игрока i, C_i — функция затрат для игрока i.

Ситуацию $u^* = (u_1^*, u_2^*, \dots, u_n^*)$, будем называть равновесием по Нэшу, если выполняется:

$$C_i(u_1^*, \dots, u_{i-1}^*, u_i^*, u_{i+1}^*, \dots, u_n^*) \leq C_i(u_1^*, \dots, u_{i-1}^*, u_i^*, u_{i+1}^*, \dots, u_n^*),$$

где $u_i \in U_i$, $i \in N$.

Для исследования зададим множество игроков $N = \{1, \dots, n\}$. Для каждого игрока i существует непустое множество M_iагентов, расположенных в подпространстве

$$W = \{1, \dots, W_x\} \times \{1, \dots, W_y\} \subset R^2,$$

в узлах целочисленной решетки, где W_x, $W_y \in N$. Агенты разных игроков могут находиться в одной позиции. Координаты каждого агента описываются в ДПСК. Для изображения агентов будем пользоваться точками на плоскости. Между агентами v_p^i и v_s^i, $s \neq p$, игрока i есть устойчивая связь (v_p^i, v_s^i), при условии их соседнего расположения в целочисленной решётке. Если между точками существует связь, то её обозначаем непрерывным отрезком.

Для игрока i задаётся вероятность p_s^i функционирования агента v_s^i. Предполагается, что эта вероятность одинакова в любой момент времени $\tau_j, j = \{1, 2, \dots\}$. Ввиду этого, у всех агентов, которые могут выйти из строя, определена функция состояния

$$I_s^i(\tau_j) = \begin{cases} 1, & v_s^i \text{ в строю} \\ 0, & v_s^i \text{ не в строю} \end{cases}$$

В любой момент времени $\tau_j, j = 1, 2, \ldots$ множество агентов (точек) $v_s^i \in M_i$, $i = 1, \ldots, n$, и связей (отрезков) $e = (v_p^i, v_s^i)$, $v_p^i \in M_i$, $v_s^i \in M_i$, $s \neq p$, $i = 1, \ldots, n$, определяет граф $R_i = (V_i, E_i)$, где V_i— непустое конечное множество элементов, называемых вершинами (или точками), E_i — конечное множество неупорядоченных пар элементов из V_i, называемых ребрами (или линиями).

Между элементами разных графов не будет существовать связей. Это обеспечит то, что неподвижные агенты одного игрока не смогут взаимодействовать с неподвижными агентами других игроков в процессе передачи информации.

Путем в графе R_i будем называть такую последовательность $e = (e_1, \ldots, e_k, \ldots)$ ребер, что конец каждого предыдущего ребра совпадает с началом следующего. Длина пути $e = (e_1, \ldots, e_k)$ — это число k рёбер последовательности. Длиной одного ребра будем считать единицу. В дальнейшем нас будет интересовать путь из одной фиксированной вершины v_s^i в другую v_t^i. Будем использовать $e(R_i)$, подразумевая путь от v_s^i до v_t^i.

1.2. Неподвижные агенты: концевые вершины

Для каждого игрока существует пара вершин v_s^i и v_t^i, между которыми необходимо установить соединение. Свойство данных агентов — это абсолютная работоспособность, т.е. вероятность их работы в каждый момент равна 1. Создание кратчайшего пути между данными вершинами является целью проводимого исследования.

1.3. Неподвижные агенты: промежуточные вершины

У каждого игрока имеются в наличии агенты V, которые обеспечивают промежуточную связь. Их особенностью является наличие у каждого такого агента величины $p \in (0,1)$, отвечающей за функционирование агента. Если при розыгрыше случайной величины полученное значение $value \in (p, 1)$, то данный агент выходит из строя, то есть теряется возможность его использования. Но спустя количество ходов t, данный агент

восстанавливает возможность участия в построении пути. Стоит обратить внимание, что если промежуточный агент является шарнирной вершиной, то, при выходе его из строя, путь между концевыми вершинами пропадает. Если у игрока большое количество шарнирных вершин, с небольшой надежностью, то возможна ситуация, что необходимый путь между концевыми вершинами не будет существовать. При этом существующего количества подвижных дронов может не хватить на поддержку существования пути.

1.4. Подвижные агенты: дроны

У каждого игрока имеется некоторое количество подвижных агентов. В рамках данного исследования их два. Подвижные дроны — это агенты, положение которых на прямоугольной решетке может меняться в процессе игры. Связь подвижного агента с любым другим агентом устанавливается аналогично связи существующей между неподвижными агентами. При этом подвижный агент может устанавливать связи с каждым работающим агентом любого игрока. На начало игры все подвижные дроны расположены в такой позиции, что их расположение не влияет на начальную ситуацию. В данном исследовании подвижные агенты не могут выйти из строя.

С помощью подвижных дронов игроки стремятся сократить затраты на своём результирующем графе $R_i^* = (V_i^*, E_i^*)$. V_i^* — это концевые и работающие промежуточные вершины текущего игрока плюс подвижные агенты всех игроков. E_i^* — это связи между соседними вершинами в V_i^*. Через $e(R_i^*)$ будем обозначать длину пути между v_s^i и v_t^i на таком графе.

1.5. Альтернативы

Задачей каждого игрока является создание кратчайшего пути между концевыми вершинами. Для этого они располагают своих подвижных агентов так, чтобы минимизировать, либо создать путь на результирующем подграфе.

Игроки ходят по очереди. На каждом шаге игрок i может совершить выбор одной из альтернатив W_i^*. Множеством альтернатив W_i^* для игрока i являются те возможные позиции на прямоугольной решётке, подстановка в которые подвижного агента создает или уменьшает, относительно текущего положения, путь от стартовой вершины до конечной. Множество альтернатив для игрока i на каждом шаге игры определяется его предыдущими ходами, и предыдущими ходами других игроков.

Каждый игрок i на каждом шаге решает задачу оптимизации следующего вида:

$$\bar{q}_i = argmin\, e(R_i), q_i \in W_i^*,$$

где $e(R_i)$ — длина пути между концевыми вершинами.

Игра продолжается пока хотя бы для одного игрока существуют альтернативы. Если альтернатив нет, а у каждого игрока есть путь между концевыми вершинами, то игра завершается. Если хотя бы у одного игрока путь не существует, то достигнута ситуация, в которой не существует равновесие по Нэшу. Функцию затрат i-го игрока будем вычислять следующим образом:

$$C_i = e(R_i^*) \geq 0.$$

Таким образом, задачей каждого игрока является минимизация длины пути между концевыми вершинами для расширенного графа R_i^* после применения дронов.

Данная задача может не иметь решения, при отсутствии позиции для перемещения подвижного дрона и невозможности пути между концевыми вершинами.

Глава 2. Описание программного обеспечения

2.1. Постановка задачи программного обеспечения

Для данного исследования должно быть представлено программное обеспечение, позволяющее исследовать игру. Реализация, при этом, должна удовлетворять следующим требованиям:

1. Необходимо наличие алгоритма, который на основе имеющихся данных, реализует построение сценариев игры.
2. Программа должна отвечать требованиям масштабируемости, то есть иметь возможность изменить объем поля игры, количество игроков, количество подвижных агентов.

2.2. Описание архитектуры программы

Данный программный продукт реализован на объектно-ориентированном языке программирования $C++$ в среде разработки $Visual\ Studio$ 2017.

Были использованы стандартные библиотеки: $iostream$ для работы с потоковыми данными, и $fstream$ для работы с файлами, источниками информации.

Для программы реализована функция $moveagent$, включающая в себя поиск позиции для подвижного агента и позволяющая либо перемещать его, либо оповещать о существовании лучшей позиции. Эта функция поиска позиции для агента обращается к функции $algo$. Здесь происходит преобразование данных в обрабатываемый для волнового алгоритма вид. $Algo$, в свою очередь, обращается к функции $wave$, где данный алгоритм реализован.

В программе были использованы небезопасные глобальные переменные для работы с массивами позиций подвижных дронов, что

потребовало аккуратной работы с потоками данных ввиду возможных перегрузок.

2.3. Описание реализации

Проведем вербальное описание программного продукта.

В файлах каждого игрока "$first.txt$", "$second.txt$" и "$third.txt$" находится информация о положении агентов в двумерном пространстве на момент начала игры. Эти данные представляют собой числовые значения p, соответствующие агентам: $p = 1$ для концевых агентов, $0 < p < 1$ для промежуточных V и $p = 0$ для позиции без агентов. Имеется фиксированное количество ходов $reliable$, через которое промежуточные агенты могут вернуться в строй. На начало игры для каждого игрока имеется пара агентов Q в позициях, не влияющих на ситуацию. Оба агента обладают 100% прочностью.

Массивы подвижных игроков $movingagentsfirst$, $movingagentssecond$ и $movingagentsthird$ являются глобальными переменными, что дает возможность работать с ними на протяжении всей программы.

Поиск позиции для подвижного дрона. Задаем процедуру $moveagent$ (массив позиции $positions$, номер игрока $numberplayer$, логическая переменная $check$). По номеру игрока определяем, с каким массивом подвижных агентов будем работать.

В массиве позиции текущего игрока определяем пару вершин с $p = 1$ (Такое может быть только у конечных значений необходимого пути). Заносим позиции этих агентов по осям координат в вектора $endpathhoriz$ и $endpathvert$. В массиве позиции $positions$ суммируем информацию об агентах, которые переместить невозможно, но которые оказывают влияние на ситуацию, с подвижными агентами до перемещения. В данном массиве

определяем существование пути $leng$ из одной концевой вершины в другую с помощью волнового алгоритма.

В массиве подвижных агентов для текущего игрока $movingagents$ определяем позиции его агентов $agentpositver$ и $agentposithoriz$. Проведем анализ возможного перемещения одного из агентов. Каждый агент поочередно перемещаем с его позиции на каждую перспективную. Перспективной позицией является та, что имеет больше одного соседа из массива позиции. Это дает возможность того, что в данной позиции агент может оказать влияние на ситуацию. В такой позиции определяем длину получающегося пути $lengchange$ с помощью волнового алгоритма. Если такой путь можно построить, и он меньше существующего, то запоминаем номер агента, который можно переместить $agent$ и его позиции $agentnewpositvert$ и $agentnewpositHoriz$, а также длину наилучшего пути $bestleng$.

Если значение логической переменной $check = true$, то проводившийся анализ был направлен на поиск возможных позиций для улучшения ситуации. Это необходимо для определения существования равновесия по Нэшу.

Если $check = false$, то процедура была направлена на поиск наилучшей позиции для перемещения агента. Если $bestleng$ существует, то по полученным данным перемещаем выбранный агент в его новую позицию и заносим это в глобальный массив позиции подвижных агентов соответствующего игрока.

Основное тело программы. Начинаем с заполнения двумерных массивов позиции для каждого игрока $firstplayerposit$, $secondplayerposit$ и $thirdplayerposit$ значениями из файлов. На данном этапе задаются массивы $firstcheck$, $secondcheck$ и $thirdcheck$ проверки состояния работоспособности и соответствующие им массивы времени нахождения вне

строя $firstcheckcount$, $secondcheckcount$ и $thirdcheckcount$. На начало игры все значения в первом массиве будут $true$, а во втором 0.

Выполняем проверку существования позиции для улучшения ситуации на начало игры. Для этого по каждому игроку проводим процедуру $moveagent$ со значением $check = true$. Если хотя бы для одного игрока можно улучшить ситуацию, то приступаем к игре. Рассмотрим её на примере хода первого игрока. Если позиция неподвижного агента $firstcheck = false$, но счетчик выхода из строя достиг $reliable$, то данный агент возвращается в работу, а соответствующая ему позиция в $firstcheckcount$ обнуляется. Если позиция недоступна, но значение меньше необходимого, то счетчик увеличивается на $+1$. Разыгрываем случайную величину $randvalue$ для каждой позиции неподвижного агента. Если значение $randvalue \in (p, 1)$, а вершина находится в строю $firstcheck = true$, то меняем соответствующие значения на $firstcheck = false$, $firstcheckcount = 1$. Если значение $randvalue \in (0, p)$ и агент находится в строю, то заносим его в массив доступных позиций $firstplayeravailable$. Если значение агента $p = 1$, то его также заносим в $firstplayeravailable$. Проводим процедуру $moveagent$ при $check = false$, что означает поиск наилучшей позиции и перемещение туда агента. В эту процедуру мы заносим массив доступных позиций $firstplayeravailable$. Для других игроков все аналогично.

После завершения процедуры последним игроком, выполняем проверку существования позиции для улучшения ситуации на массивах с уже перемещёнными подвижными агентами. Если хотя бы для одного игрока можно улучшить положение, то переходим к следующему ходу.

Если после завершения перемещений подвижных агентов всеми игроками доступных позиций для улучшения ситуации нет, то проводим процедуру проверки наличия путей $checkpath$ в массивах позиций текущих ситуаций. Данная проверка также происходит с помощью волнового

алгоритма. Если хотя бы у одного игрока отсутствует путь между концевыми вершинами, то достигнут крайний случай. Такая ситуация означает проигрыш.

Если для всех игроков перемещение агентов является невозможным и при этом у всех существует путь между концевыми вершинами, то получена ситуация, удовлетворяющая равновесию по Нэшу.

2.4. Описание волнового алгоритма

Данный алгоритм работает с планарным графом $G = (V, E)$, отображенным на двумерную плоскость. Визуализация графа будет представлять собой ограниченную замкнутой линией решетку. На пересечении кривых этой решетки будут возможные позиции агентов, а сами кривые будут обозначать связи между агентами. Все позиции можно интерпретировать в двух видах: проходимые, которые при поиске пути можно исследовать, и непроходимые, путь через которые запрещён, стартовая позиция v_s^i и финишная v_t^i. В контексте работы стартовая и финишная позиции — это концевые агенты, проходимые позиции — это промежуточные агенты, находящиеся в строю, и подвижные дроны всех участвующих игроков. Соответственно, непроходимые – это позиции без дронов либо с нефункционирующими агентами. Между стартовой и финишной позицией будет искаться кратчайший путь.

Алгоритм предназначен для поиска кратчайшего пути от стартовой позиции v_s^i к конечной позиции v_t^i. При отсутствии пути будет выдана соответствующая метка [13].

Перед началом алгоритма строится образ всех позиций обрабатываемого поля: для каждой позиции приписывается свойство проходимости, запоминаются стартовая и финишная позиции.

Соседними позициями будут считаться те, которые расположены рядом по вертикали либо по горизонтали.

Если вершина проходима и не помечена, то её значение становится равно количеству шагов от стартовой вершины. Для стартовой позиции это значение всегда будет 1.

1. Начальное состояние. Значения всех участвующих позиций с агентами в поиске пути имеют значение 0. Позиции непроходимые либо с вышедшими из строя агентами имеют значение -1.

2. Шаг алгоритма. Если все возможные вершины при переходе из v_s^i пройдены, то алгоритм завершается со значением -1 по причине отсутствия пути в v_t^i. Если достигнуто v_t^i, то возвращается значение расстояния до v_t^i. В противном случае, для рассматриваемой вершины все соседние позиции со значениями 0 получают метку, увеличенную на единицу по сравнению с рассматриваемой вершиной. Когда все соседи у рассматриваемой вершины помечены, совершается переход в соседнюю вершину с большим значением метки. Если таких вершин нет, то происходит возвращение на вершину с меньшей меткой. Повторение шага алгоритма [14].

Глава 3. Тестирование

3.1. Постановка игры.

Тестирование программы проведено на примере игры трёх лиц , $N = \{1,2,3\}$. Сетевые структуры для каждого игрока, рис. 1, рис. 2, рис. 3, хранятся в соответствующих файлах. В начале игры у всех игроков существуют пути между концевыми вершинами v_s^i и v_t^i, и их длины равны $l_1 = 17$, $l_2 = -1$, $l_3 = 21$. Целью игроков является создание пути с минимальной длиной.

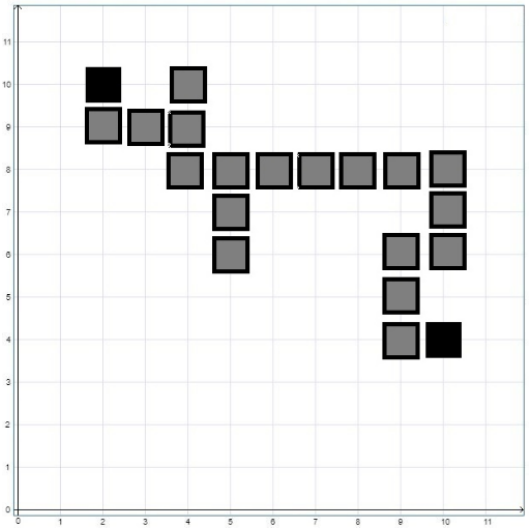

Рис. 1: Сетевая структура первого игрока до игры.

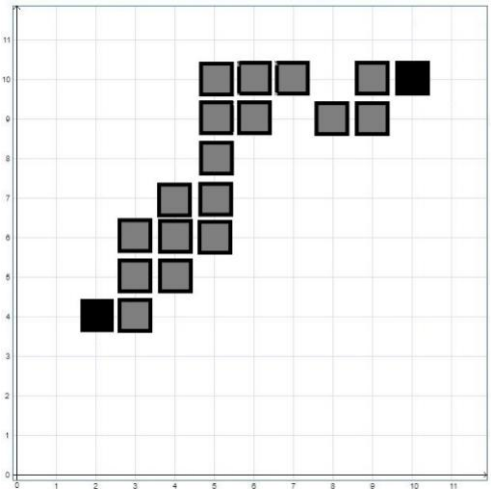

Рис. 2: Сетевая структура второго игрока до игры.

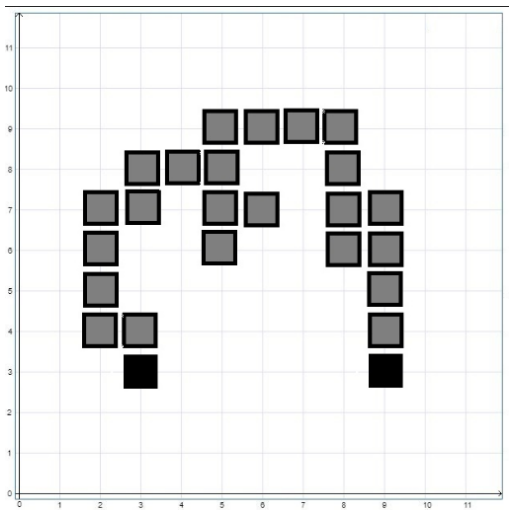

Рис. 3: Сетевая структура третьего игрока до игры.

3.2. Благоприятный исход.

Приведена ситуация, когда игра завершается ситуацией равновесия по Нэшу. Ход игры изображен на рис. 4.

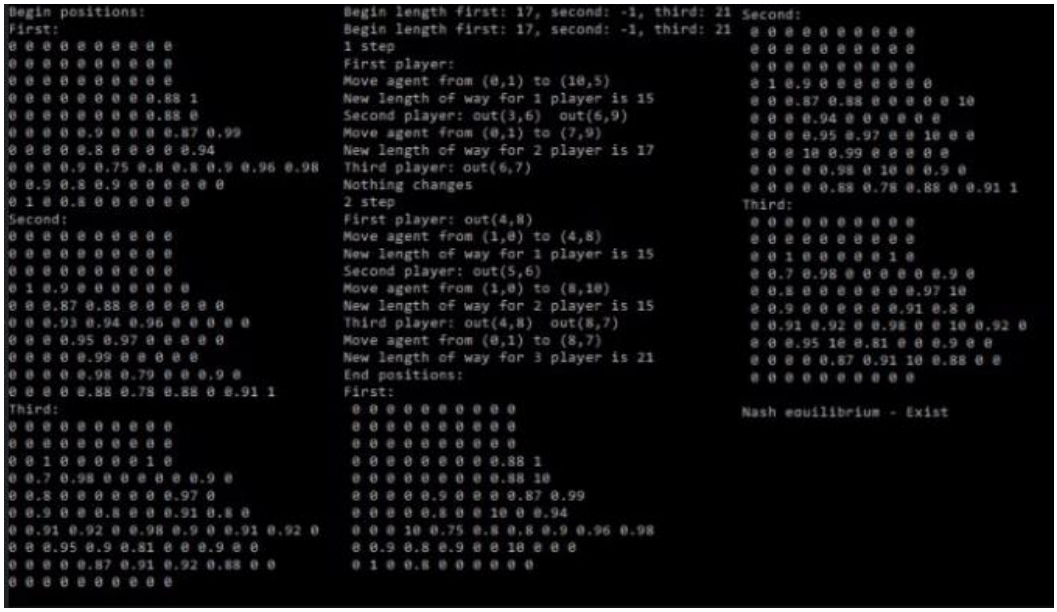

Рис. 4: Результат тестирования, благоприятный исход.

3.3. Неблагоприятный исход.

Приведена игра с аналогичными начальными положениями, как в случае благоприятного исхода, но с меньшими значениями p у неподвижных

агентов. Игра завершается ситуацией крайнего случая. Ход игры представлен на рис. 5.

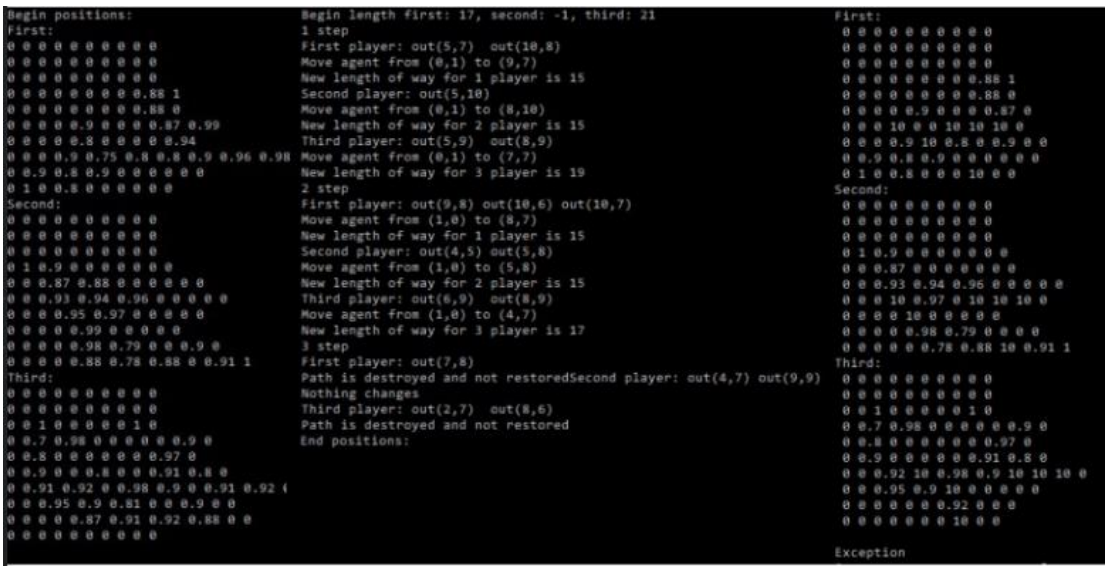

Рис. 5: Результат тестирования, крайний случай.

Глава 4. Подробные примеры игры

4.1. Игра с благоприятным исходом.

Рассмотрим пример данной игры при ситуациях, когда мы можем достичь благоприятного исхода.

Игра Г ведется для игроков, где $N = \{1,2,3\}$. Для каждого игрока игра будет происходить на отдельных сетевых структурах, представленных на рис. 6, рис. 7, рис. 8. Между элементами структур разных игроков не может быть взаимосвязи, кроме связей с множествами подвижных агентов Q_i, $i = \{1,2,3\}$. У игрока будет свое, подконтрольное ему, множество подвижных агентов, но свое влияние они будут оказывать на все сетевые структуры. На рис. 9 представлены обозначения подвижных агентов для каждого игрока.

Положение агентов в структуре задается координатами в декартовой прямоугольной системе координат (ДПСК). Связь между агентами существует в случае, если они расположены по соседству по вертикали либо по горизонтали.

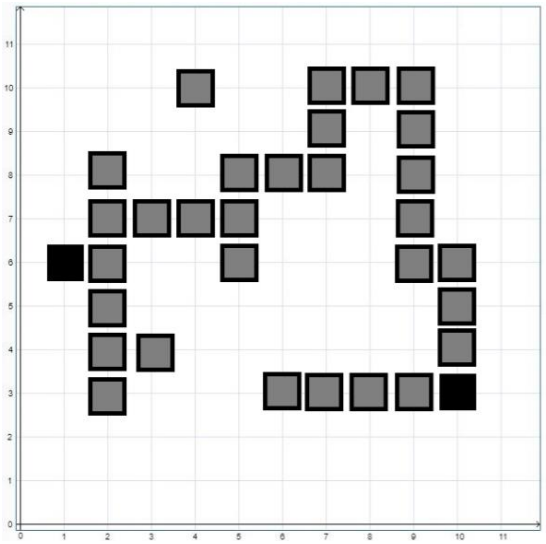

Рис. 6: Сетевая структура первого игрока до игры.

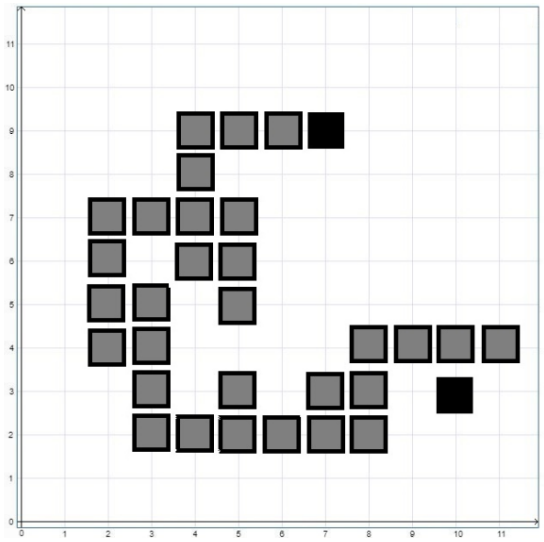

Рис. 7: Сетевая структура второго игрока до игры.

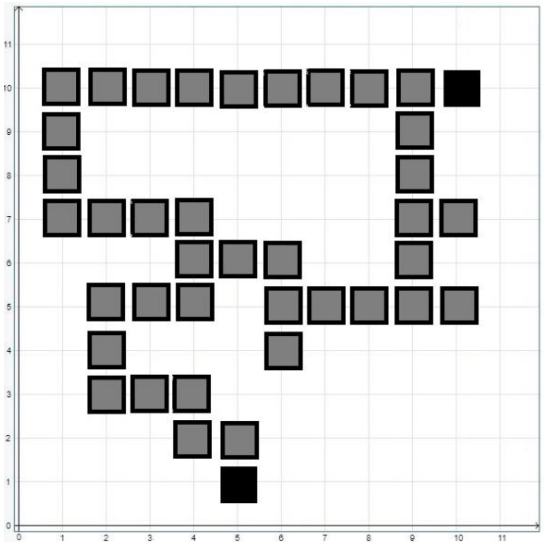

Рис. 8: Сетевая структура третьего игрока до игры.

Рис. 9: Изображения подвижных агентов для соответствующих игроков.

Начинаем с того, что определяем существование первоначального пути либо его отсутствие на первоначальном графе, рис. 10, рис. 11, рис. 12. Длина пути между v_s^i и v_t^i для первого игрока $l_1 = 19$, для второго $l_2 = 22$, для третьего $l_3 = 21$. Определим, существуют ли возможные позиции

для улучшения текущих путей подвижными агентами. Как видно, для первого игрока это позиции (8,8) и (8,9); для второго – (3,6), (4,5) и (5,4); для третьего – (3,4), (4,4) и (5,5).

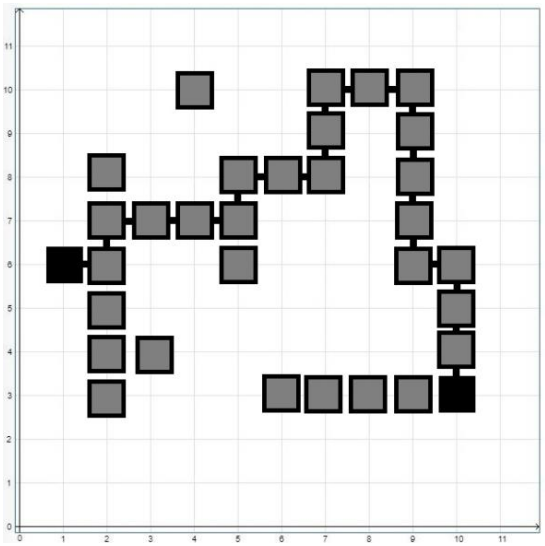

Рис. 10: Первоначальный путь для первого игрока

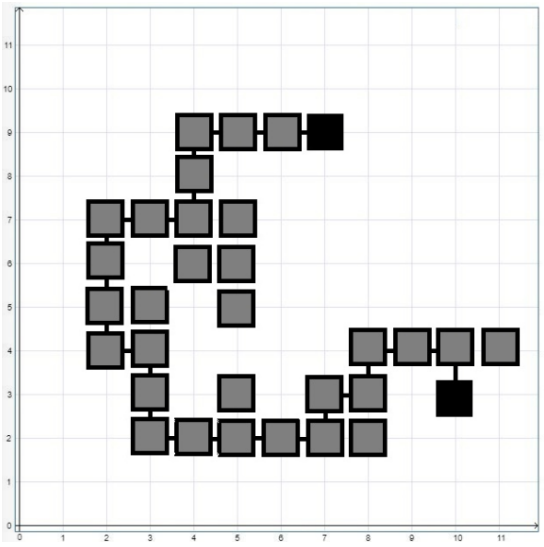

Рис. 11: Первоначальный путь для второго игрока

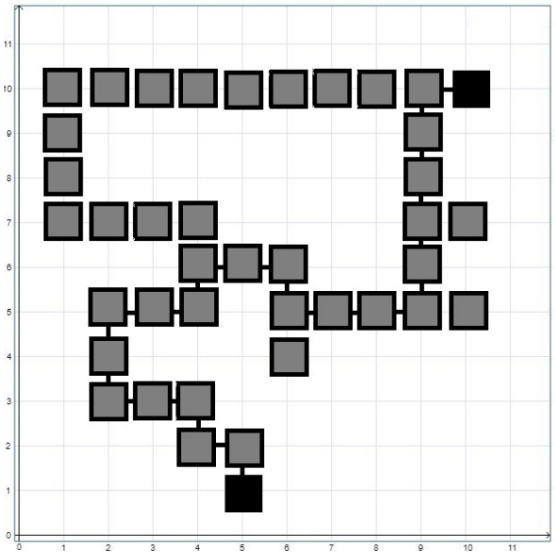

Рис. 12: Первоначальный путь для третьего игрока

Так как возможные позиции для улучшения путей существуют, то начинаем разыгрывать случайную величину $randvalue$. Соответствующие агенты, вышедшие из строя, теряют свою окраску на изображениях, а также помечаются метками. Эти метки определяют количество ходов, проводимых агентами вне строя, при достижении лимита $reliable$ они возвращаются в структуру. Разыгрываем случайную величину для первого игрока $randvalue$, получаем структуру на рис. 13.

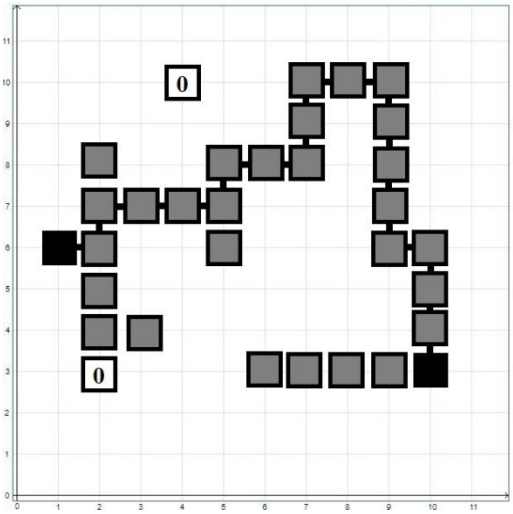

Рис. 13: Ситуация для первого игрока после розыгрыша случайной величины, 1 ход.

На существующий путь вышедшие из строя агенты никак не повлияли, поэтому для агента выбираем позицию, уменьшающую путь. Самой наилучшей позицией будет (8,8). Помещаем агента туда, рис. 14. Получаем $l_1 = 15$.

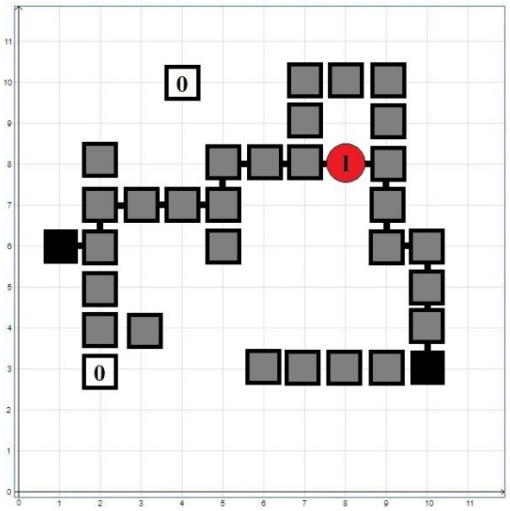

Рис. 14: Ситуация для первого игрока после перемещения своего агента, 1 ход.

Теперь разыгрываем случайную величину $randvalue$ для второго игрока, рис. 15. Заметим, что перемещенный агент первого игрока уже может оказывать влияние на структуру второго игрока.

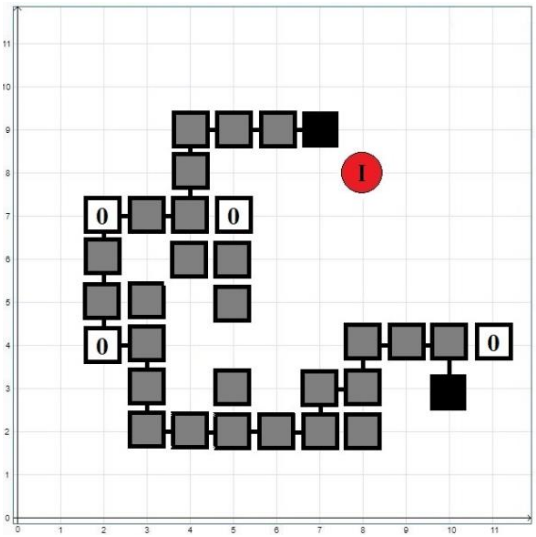

Рис. 15: Ситуация для второго игрока после розыгрыша случайной величины, 1 ход.

Из строя вышли пара вершин, участвовавших в существующем пути. Тем не менее, можно, не заменяя этих агентов, переместить дрон на позицию (5,4), создавая новый оптимальный путь, рис. 16. Несмотря на свое присутствие, дрон первого игрока на ситуацию не влияет. Получаем $l_2 = 18$.

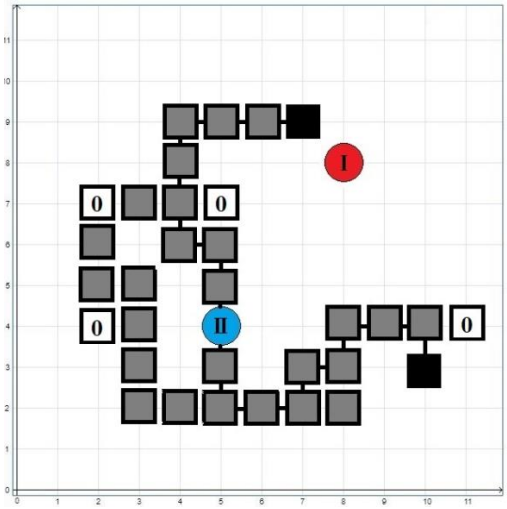

Рис. 16: Ситуация для второго игрока после перемещения своего агента, 1 ход.

Разыгрываем случайную величину $randvalue$ для третьего игрока, рис. 17. Перемещенные агенты первого и второго игроков уже оказывают влияние на сеть.

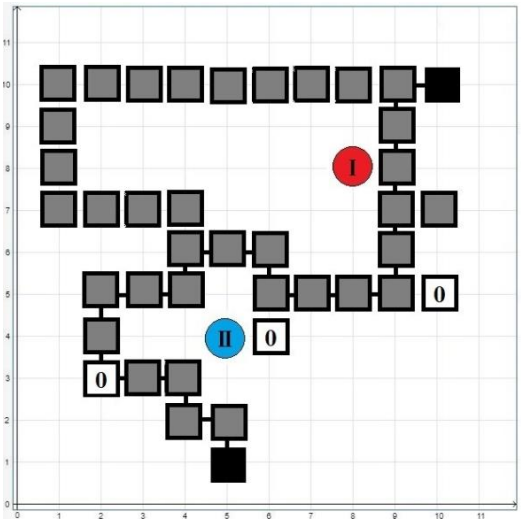

Рис. 17: Ситуация для третьего игрока после розыгрыша случайной величины, 1 ход.

Из строя вышел агент, участвующий в пути. Однако позиция (4,4) позволяет создать путь, рис. 18, который одновременно короче первоначального и обходит поврежденный участок. Теперь $l_3 = 17$.

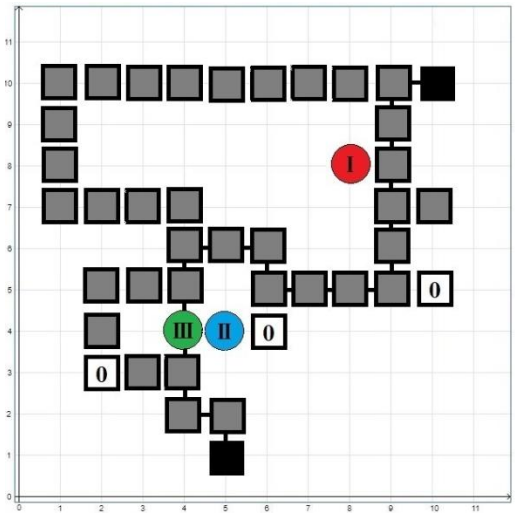

Рис. 18: Ситуация для третьего игрока после перемещения своего агента, 1 ход.

1 ход закончился. Рассмотрим ситуацию для первого игрока, рис. 19, для второго, рис. 20, и для третьего, рис. 18. Заметим, что для первого и второго игроков в структуру включается агент третьего игрока.

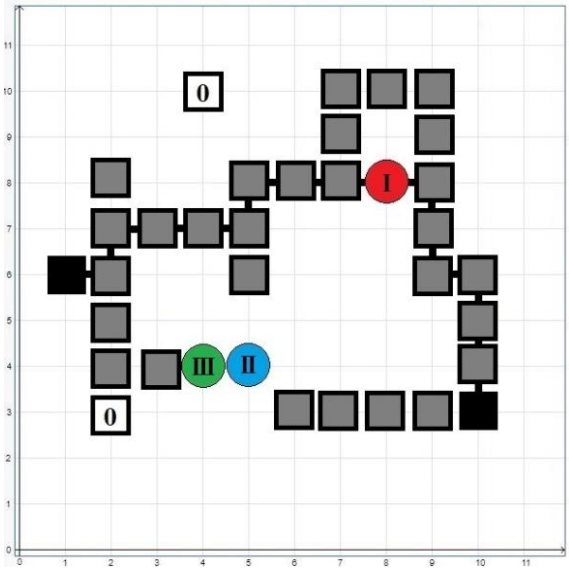

Рис. 19: Ситуация для первого игрока после первого хода.

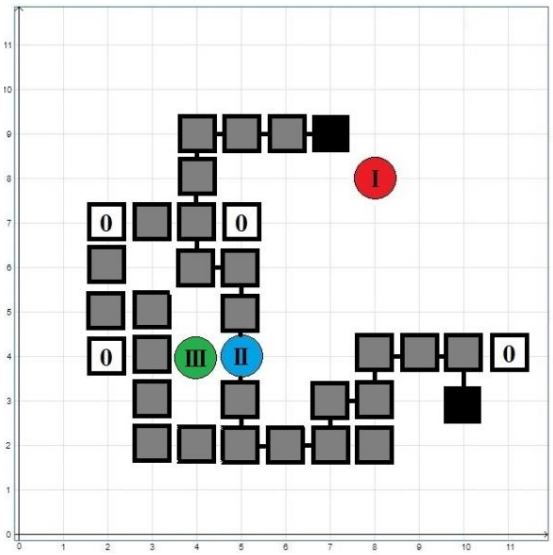

Рис. 20: Ситуация для второго игрока после первого хода.

Ввиду наличия дронов других игроков, для первого игрока существуют позиции (5,3) и (6,4), где, используя подвижные агенты второго и третьего игроков, можно сократить путь. Для второго игрока существует позиция (6,3), позволяющая сократить путь. Для третьего – имеется позиция (5,5), поставив агент на которую, можно сократить путь. Таким образом, начинаем второй ход.

Для начала опять разыгрываем случайную величину для неподвижных агентов первого игрока. Метки на агентах, которые вышли из строя на предыдущем ходу, увеличиваются, рис. 21.

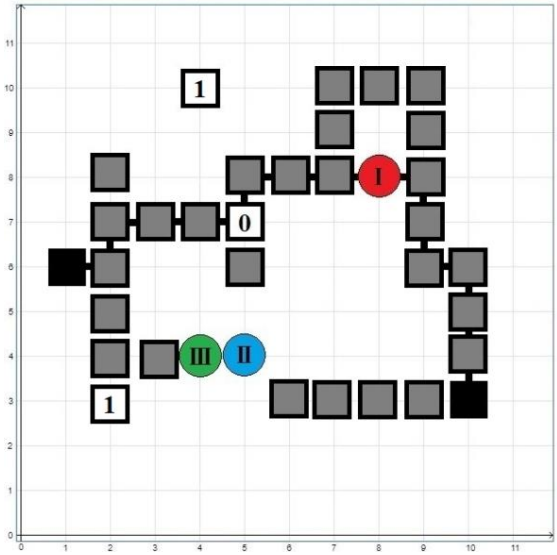

Рис. 21: Ситуация для первого игрока после розыгрыша случайной величины, 2 ход.

Вышел из строя агент на основном пути. Однако позиции агентов других игроков достаточно благоприятны и позволяют создать путь, который короче текущего и позволяет обойти поврежденный участок. Задействуем неиспользованный агент, рис. 22. Текущий $l_1 = 11$.

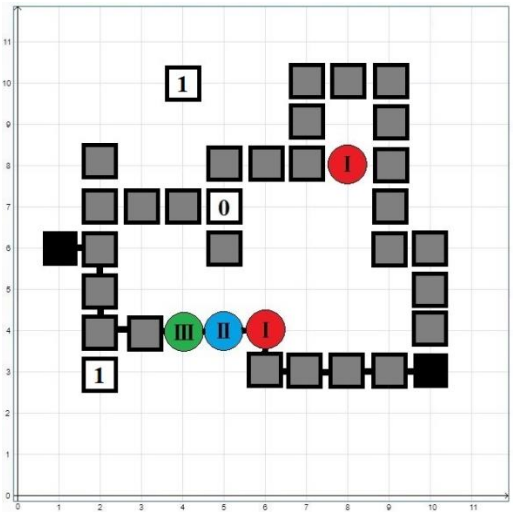

Рис. 22: Ситуация для первого игрока после перемещения своего агента, 2 ход.

Разыгрываем случайную величину для агентов второго игрока. Также увеличиваем метки для поврежденных ранее агентов, рис. 23.

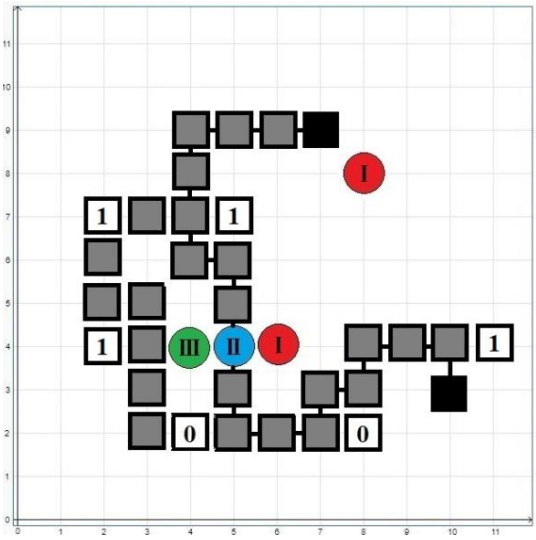

Рис. 23: Ситуация для второго игрока после розыгрыша случайной величины, 2 ход.

Вышедшие из строя агенты не оказывают влияние на путь, однако подвижный агент первого игрока создает перспективную позицию (7,4), куда переместим незадействованного дрона, рис. 24. Полученный путь $l_2 = 14$.

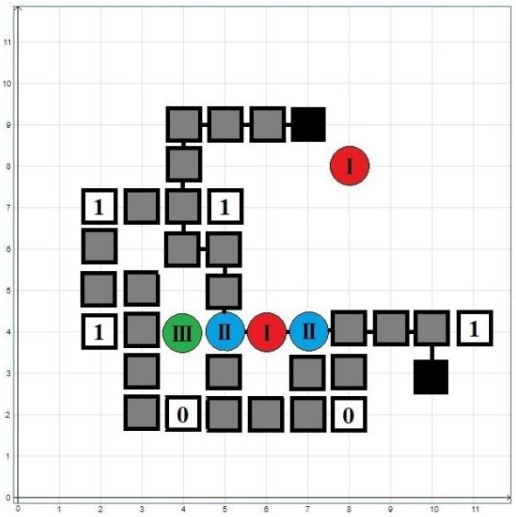

Рис. 24: Ситуация для второго игрока после перемещения своего агента, 2 ход.

Разыгрываем случайную величину для третьего игрока. Увеличиваем метки на вышедших из строя на предыдущих шагах агентах, рис. 25.

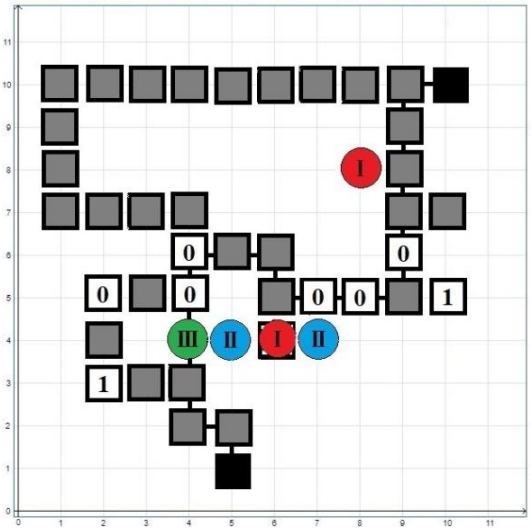

Рис. 25: Ситуация для третьего игрока после розыгрыша случайной величины, 2 ход.

На существующем пути вышло из строя большое количество агентов и путь больше не существует. Однако, переместив неиспользованного агента в позицию (4,6), можно восстановить путь, но длиннее существовавшего, рис. 26. $l_3 = 25$.

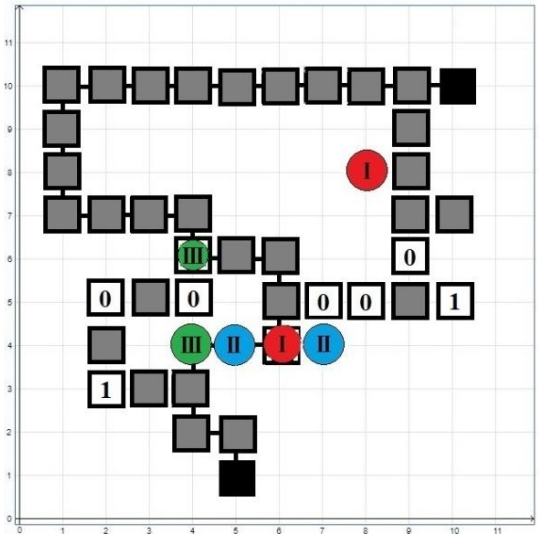

Рис. 26: Ситуация для третьего игрока после перемещения своего агента, 2 ход.

Рассмотрим получившиеся ситуации для первого игрока, рис. 27, для второго, рис. 28, и для третьего, рис. 26.

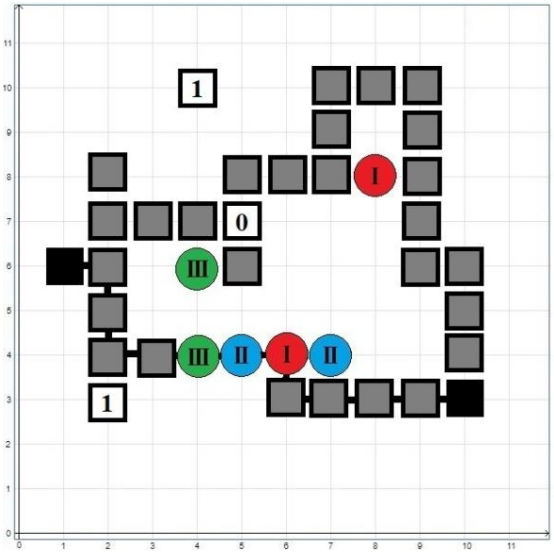

Рис. 27: Ситуация для первого игрока после второго хода.

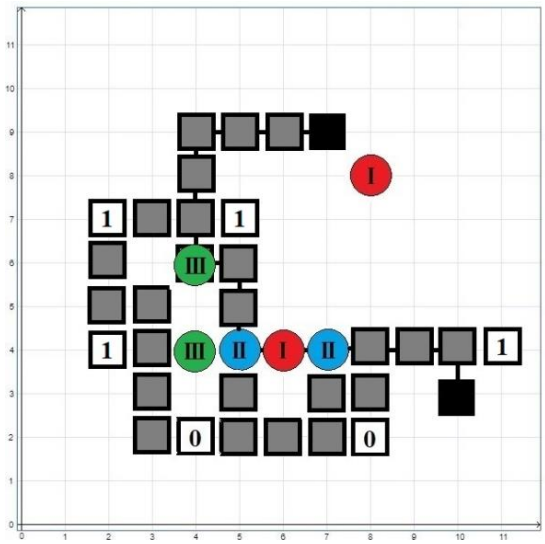

Рис. 28: Ситуация для второго игрока после второго хода.

В данных ситуациях подвижные агенты первого и второго игроков не обладают перспективными позициями для улучшения ситуации. Однако для третьего игрока есть позиция (5,3), куда он может переместить дрона с позиции (4,4), поэтому переходим на следующий ход.

Разыгрываем случайную величину для первого игрока. Увеличиваем метки для вышедших ранее из строя агентов, рис. 29.

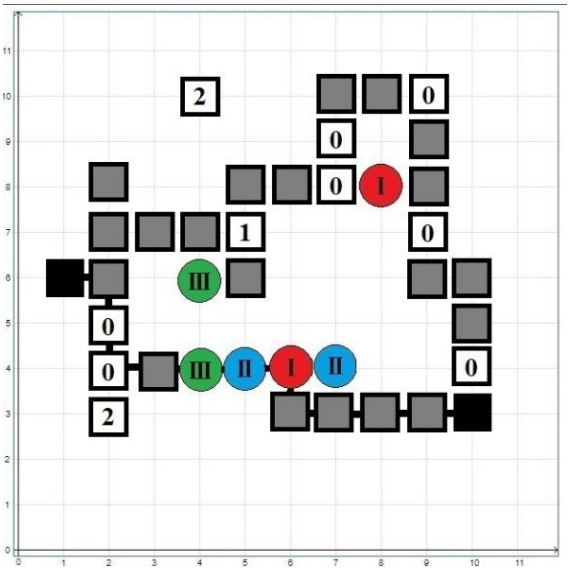

Рис. 29: Ситуация для первого игрока после розыгрыша случайной величины, 3 ход.

На текущем пути из строя вышло больше одной вершины, таким образом, задействовав неиспользуемый агент с позиции (8,8), восстановить путь не получится. Следовательно, необходимо строить новый путь, что позволяет позиция (4,5), рис. 30. $l_1 = 13$.

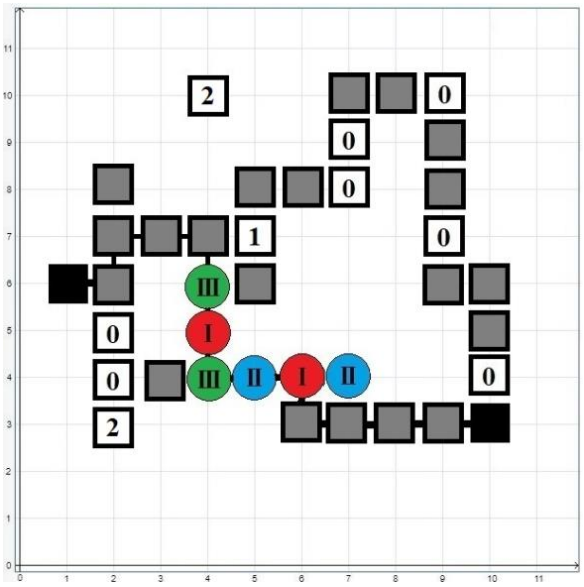

Рис. 30: Ситуация для первого игрока после перемещения своего агента, 3 ход.

Разыгрываем случайную величину для второго игрока. Соответственно, повышаем значение на отмеченных вершинах, ранее вышедших из строя, рис. 31.

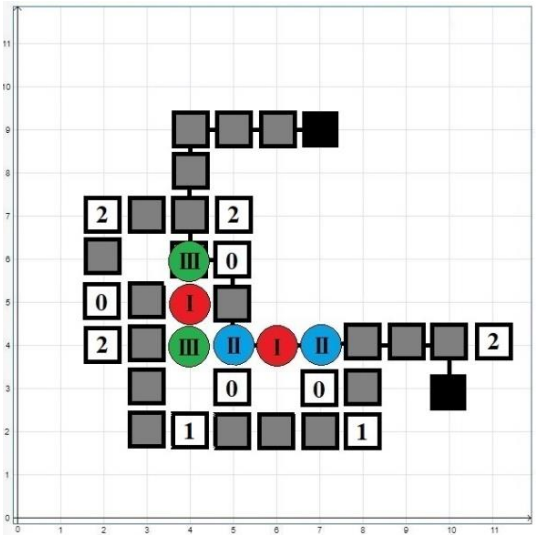

Рис. 31: Ситуация для второго игрока после розыгрыша случайной величины, 3 ход.

Вышел из строя неподвижный агент, участвующий в пути. Однако положение подвижных агентов других игроков позволяет восстановить путь без перемещения подвижных агентов, сохраняя $l_2 = 14$, рис. 32.

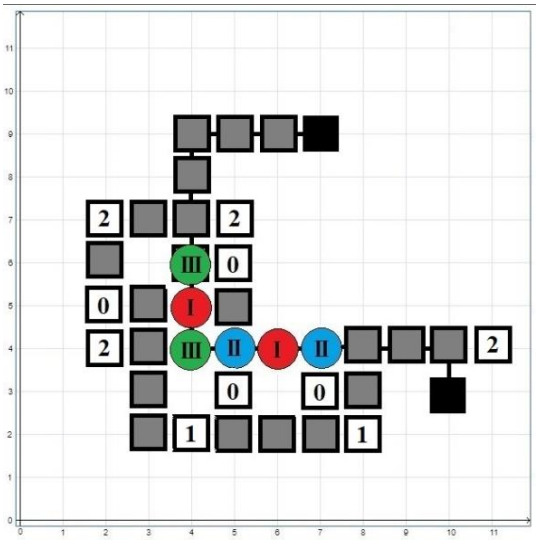

Рис. 32: Ситуация для второго игрока после перестройки своего пути, 3 ход.

Разыгрываем случайную величину для подвижных агентов третьего игрока. Аналогично увеличиваем метки, рис. 33.

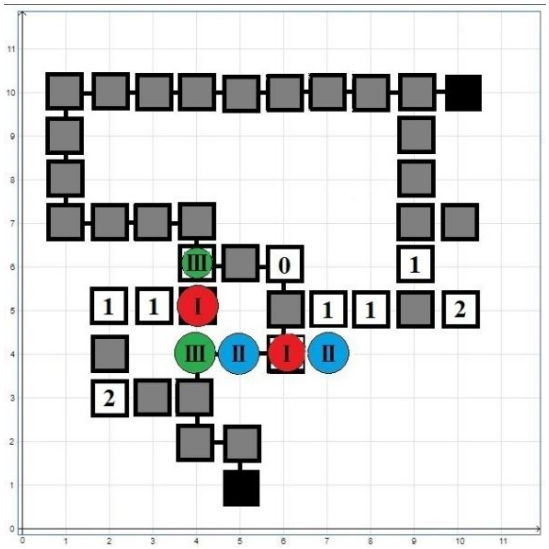

Рис. 33: Ситуация для третьего игрока после розыгрыша случайной величины, 3 ход.

Из строя вышел неподвижный агент, участвующий в пути, однако положение подвижного агента первого игрока на (4,5) позволяет перестроить путь, обходя поврежденный участок и сокращая путь без перемещения агента, рис. 34. $l_3 = 21$.

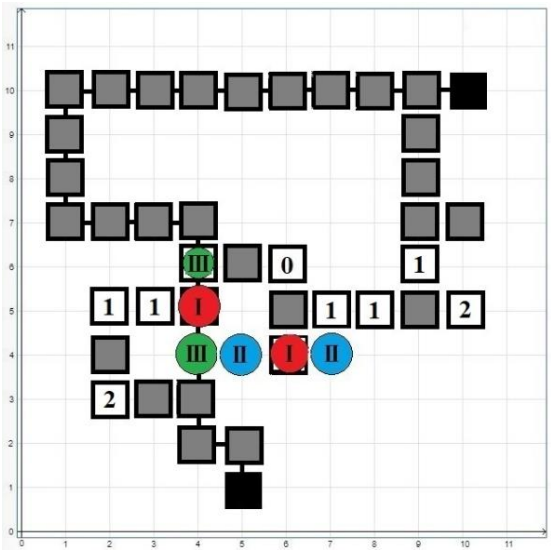

Рис. 34: Ситуация для третьего игрока после перестройки своего пути, 3 ход.

Заметим, что второй и третий игроки не перемещали своих агентов. Таким образом, конец третьего хода для каждого игрока иллюстрируют рис. 30, рис. 32 и рис. 34. Никто из игроков не может переместить своих агентов так, чтобы улучшить свое положение. Проверяем каждого игрока на существование пути для их позиции. В данном случае, у всех троих игроков есть путь из концевой вершины v_s^i в концевую вершину v_t^i. Считаем, что найдено равновесие по Нэшу, а значит, игра закончена.

4.2. Игра с неблагоприятным исходом.

Рассмотрим подобную ситуацию на примере игры с идентичными ситуациями на конце первого хода. Таким образом, будем считать текущей ситуацию для первого игрока, рис. 19, для второго, рис. 20, и для третьего, рис. 18.

Как и в предыдущем примере, разыгрываем случайную величину для неподвижных агентов первого игрока, параллельно увеличив метки на вышедших из строя агентов на первом шаге, рис. 35.

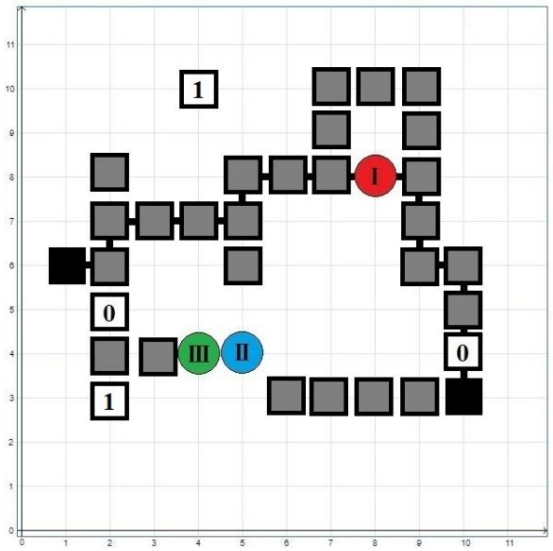

Рис. 35: Ситуация для первого игрока после розыгрыша случайной величины, 2 ход.

В данной ситуации вышли из строя агенты, один из которых находится на текущем пути, а другой позволяет построить путь с использованием подвижных агентов других игроков. Таким образом, неиспользуемый дрон можно переместить только на позицию, вышедшего из строя агента (10, 4), рис. 36. $l_1 = 15$.

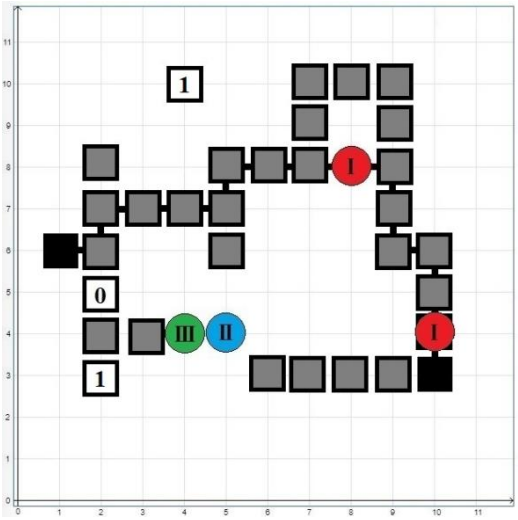

Рис. 36: Ситуация для первого игрока после перемещения своего агента, 2 ход.

Разыгрываем случайную величину для второго игрока. Повышаем метки для неработающих с прошлого хода агентов, рис. 37.

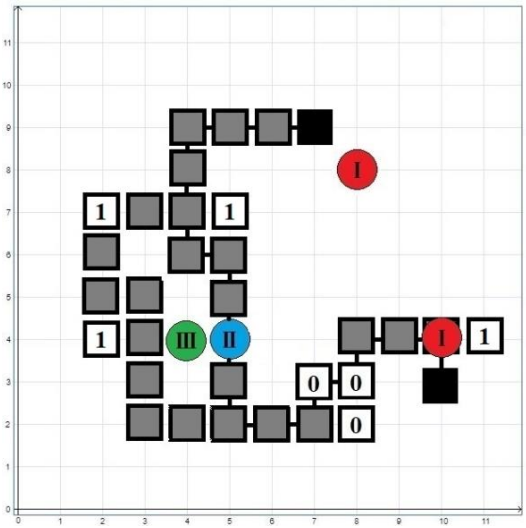

Рис. 37: Ситуация для второго игрока после розыгрыша случайной величины, 2 ход.

Из строя вышла группа агентов, лежащих на текущем пути. Используя одного неподвижного агента, восстановить путь не представляется возможным. Необходимо знать, куда поставит свой дрон третий игрок, чтобы проверить позицию его агента для создания пути. $l_2 = 0$.

Разыгрываем случайную величину для третьего игрока. Увеличиваем метки на вышедших из строя агентах на первом ходу, рис. 38.

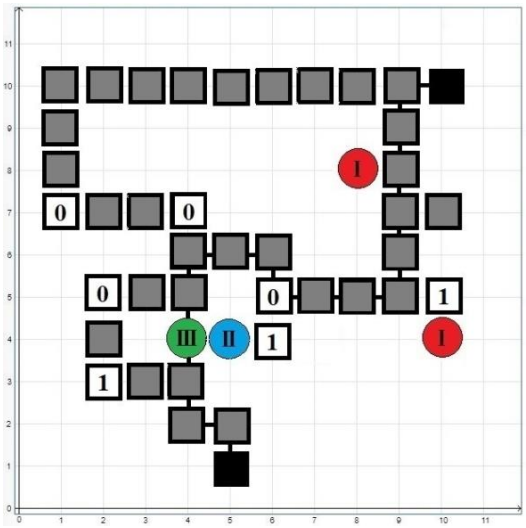

Рис. 38: Ситуация для третьего игрока после розыгрыша случайной величины, 2 ход.

Из строя вышел неподвижный агент на позиции (6,5), принадлежащий текущему пути. Неиспользовавшийся дрон отправлен туда, рис. 39. $l_3 = 17$.

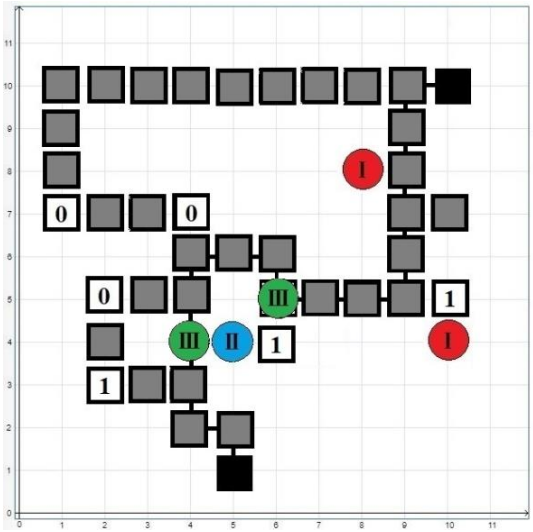

Рис. 39: Ситуация для третьего игрока после перемещения своего агента, 2 ход.

Для первого игрока текущее положение стало рис. 40, для второго рис. 41, для третьего рис. 39.

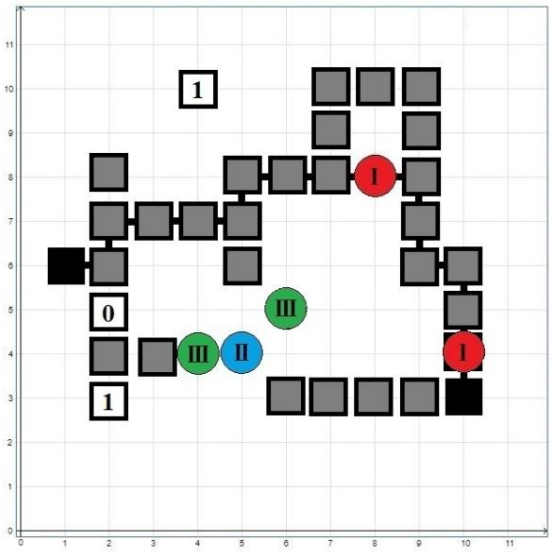

Рис. 40: Ситуация для первого игрока после второго хода.

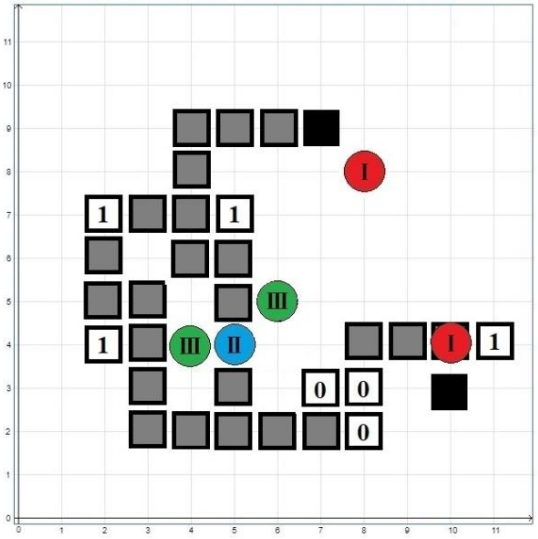

Рис. 41: Ситуация для второго игрока после второго хода.

Никто из игроков не может переместить своего агента так, чтобы улучшить свое положение. При этом для второго игрока отсутствует путь между концевыми вершинами. Наблюдается крайний случай. Игра завершается, и считается проигранной.

4.3. Выводы

Самые проблемными позициями обладают вершины, являющиеся шарнирами графа позиций. В случае выхода из строя одного неподвижного агента на такой позиции, его можно заменить подвижным дроном. Если из

строя вышло больше одного подобного агента, в течение хода восстановить путь не получится.

Вариантом решения данной проблемы можно считать подстановку на шарнирные позиции агентов с самым высоким коэффициентом надежности. Другой вариант, стараться обходиться без подобных вершин.

На стадии планирования спроектировать такую группировку неподвижных агентов, чтобы, в случае выхода из строя одного из них, связь мог поддержать его соседний без использования подвижного дрона. Можно рассматривать возможность существования альтернативных путей.

Ввиду возможности возвращения неподвижных агентов в рабочее состояние спустя некоторое время, можно использовать технические усовершенствования для сокращения такого времени.

Заключение

В процессе данной работы был предложен и реализован алгоритм поиска оптимального расположения мобильных агентов (дронов) на сети MANET, в которой некоторые агенты могут выходить из строя с некоторыми известными вероятностями. Задача была изучена как многошаговая кооперативная игра.

В процессе работы было создано программное обеспечение, позволяющее осуществить имитационное моделирование. Тестирование было проведено на примере игры трёх лиц.

Данное исследование не имеет ограничений в масштабируемости и может быть расширено под другие рамки подобных задач.

Список литературы

[1] Смирнов Д.С., Громова Е.В. Обзор MANET: история, проблемы и приложения, 2017

[2] Goyal P., Parmar V., Rishi R. Manet: vulnerabilities, challenges, attacks, application // IJCEM International Journal of Computational Engineering & Management. 2011. Т. 11. №. 2011. P. 32–37.

[3] Петросян Л. А, Зенкевич Н. А, Шевкопляс Е. В. Теория игр. СПб.: БХВ-Петербург, 2012. 424 с.

[4] Новиков Д. А. Игры и сети. Математическая теория игр и её приложения. 2010. Т. 2. Вып. 1.

[5] М. А. Булгакова, Л. А. Петросян, Кооперативные сетевые игры с попарными взаимодействиями // Математическая теория игр и ее приложения, 2015.

[6] Ekaterina Gromova, Dmitry Gromov, Nikolay Timonin, Anna Kirpichnikova, Stewart Blakeway. A dynamic game of mobile agents placement in a MANET. SIMS 2016: 2nd International Conference on Systems Informatics, Modelling and Simulation.

[7] Taisiia Plehanova, Ekaterina Gromova, Dmitry Gromov, Stewart Blakeway, Anna Kirpichnikova. The strategic placement of mobile agents on a hexagonal graph using game theory. IEEE. DOI: 10.1109/ICAT.2017.8171635

[8] Громова Е.В., Плеханова Т.М. Кооперативная динамическая игра на сети MANET, Процессы управления и устойчивость. Труды 49-й международной научной конференции аспирантов и студентов / науч. ред. тома Н. В. Смирнов.

[9] Буре В.М., Парилина Е.М. Стохастические модели передачи данных в сетях с различными топологиями // Управление большими системами. 2017. Выпуск 68. С.6-29.

[10] T. Alpcan and L.Pavel, “Nash equilibrium design and optimization,” in Proc. Of Intl. Conf. on Game Therory for Networks (GameNets), Istanbul, Turkey, May 2009

[11] Sedakov A. Weak Equilibrium in Multistage Games with a Simple Coalitional Structure. Proceedings of the Russian–Finnish Graduate School Seminar Dynamic Games and Multicriteria Optimization, 2006

[12] В. В. Мазалов, А. Н. Реттиева “Равновесие по Нэшу в задачах охраны окружающей среды”, Матем. моделирование, 18:5 (2006), 73–90

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to Algorithms. — 3rd edition. — The MIT Press, 200917)

[14] Frank Rubin The Lee path connection algorithm // IEEE Transactions on Computers. — 1974

Приложение

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

float movingagentsfirst[11][11];
float movingagentssecond[11][11];
float movingagentsthird[11][11];
int path[11][11];
int hb, vb, he, ve;
int wave(int hor, int ver, int count)
{
	if ((hor <=1) || (hor > 10) || (ver<=1) || (ver > 10))
	{
		return -1;
	};
	if (path[ver][hor] == -1)
	{
		return -1;
	}
```

```
        else
        {
            if ((hor == he) && (ver == ve))
            {
                return count;
            };
            if (path[ver + 1][hor] == 0) { path[ver + 1][hor] = count + 1; }
            else
                if (path[ver - 1][hor] == 0) { path[ver - 1][hor] = count + 1; }
                else
                    if (path[ver][hor + 1] == 0) { path[ver][hor + 1] = count +
1; }
                    else
                        if (path[ver][hor - 1] == 0) { path[ver][hor - 1] =
count + 1; }
                        else return -1;
            int r;
            if (((path[ver + 1][hor] - path[ver][hor]) == 1) && (path[ver + 1][hor]
> 0))
            {
                r = wave(hor, ver + 1, count + 1);
                if (r > 0)
```

```
                return r;
        };
        if (((path[ver - 1][hor] - path[ver][hor]) == 1)&&(path[ver - 1][hor] >
0))
        {
            r = wave(ver - 1, hor, count + 1);
            if (r > 0)
                return r;
        };
        if (((path[ver][hor + 1] - path[ver][hor]) == 1) && (path[ver][hor + 1]
> 0))
        {
            r = wave(hor + 1, ver, count + 1);
            if (r > 0)
                return r;
        };
        if (((path[ver][hor - 1] - path[ver][hor]) == 1) && (path[ver][hor - 1] >
0))
        {
            r = wave(hor - 1, ver, count + 1);
            if (r > 0)
                return r;
```

```
        };
    };
}; // realisation of Wave Algorithm


int algo(float positions[11][11], int horbeg, int vertbeg, int horend, int vertend)
{
    for (int i = 1; i <= 10; i++)
    {
        for (int j = 1; j <= 10; j++)
        {
            if (positions[i][j] > 0)
            {
                path[i][j] = 0;
            }
            else path[i][j] = -1;
        };
    };
    hb = horbeg;
    vb = vertbeg;
    he = horend;
    ve = vertend;
```

```
        path[vb][hb] = 1;
        return wave(hb, vb, 1);
}; //Marks positions for Wave algorithm
int moveagent(float positions[11][11], int numberplayer, bool check)
{
        float movingagents[11][11];
        vector<int> endpathhoriz,endpathvert,agentpositvert,agentposithoriz;
        switch (numberplayer)
        {
        case 1:
                for (int i = 0; i <= 10; i++)
                        for (int j = 0; j <= 10; j++)
                        {
                                movingagents[i][j] = movingagentsfirst[i][j];
                                if (movingagents[i][j] = 10)
                                {
                                        agentpositvert.push_back(i);
                                        agentposithoriz.push_back(j);
                                };
                                if (positions[i][j] != 1)
                                {
```

```
                    positions[i][j] = positions[i][j] +
movingagentssecond[i][j] + movingagentsthird[i][j]+ movingagentsfirst[i][j];
                    }
                    else
                    {
                        endpathvert.push_back(i);
                        endpathhoriz.push_back(j);
                    };
                };
            break;
        case 2:
            for (int i = 0; i <= 10; i++)
                for (int j = 0; j <= 10; j++)
                {
                    movingagents[i][j] = movingagentssecond[i][j];
                    if (movingagents[i][j] = 10)
                    {
                        agentpositvert.push_back(i);
                        agentposithoriz.push_back(j);
                    };
                    if (positions[i][j] != 1)
                    {
```

```
                    positions[i][j] = positions[i][j] +
movingagentsfirst[i][j] + movingagentsthird[i][j]+ movingagentssecond[i][j];
                    }
                    else
                    {
                        endpathvert.push_back(i);
                        endpathhoriz.push_back(j);
                    };
                };
            break;
        case 3:
            for (int i = 0; i <= 10; i++)
                for (int j = 0; j <= 10; j++)
                {
                    movingagents[i][j] = movingagentsthird[i][j];
                    if (movingagents[i][j] = 10)
                    {
                        agentpositvert.push_back(i);
                        agentposithoriz.push_back(j);
                    };
                    if (positions[i][j] != 1)
                    {
```

```
                    positions[i][j] = positions[i][j] + movingagentsfirst[i][j] + movingagentssecond[i][j] + movingagentsthird[i][j];
                }
                else
                {
                    endpathvert.push_back(i);
                    endpathhoriz.push_back(j);
                };
            };
        break;
    };
    int leng = algo(positions, endpathhoriz[0], endpathvert[0], endpathhoriz[1], endpathvert[1]);
    int lengchange,agent=-1,agentnewpositvert=0,agentnewposithoriz=0,bestleng=100;
    for (int x = 0; x <= 1; x++)
    {
        movingagents[agentpositvert[x]][agentposithoriz[x]] = 0;
        positions[agentpositvert[x]][agentposithoriz[x]] = 0;
        for (int i=0;i<=10;i++)
            for (int j = 0; j <= 10; j++)
            {
```

```
if (positions[i][j] == 0)

    if (((positions[i + 1][j] > 0) && (positions[i - 1][j] > 0)) || ((positions[i + 1][j] > 0) && (positions[i][j + 1] > 0)) || ((positions[i + 1][j] > 0) && (positions[i][j - 1] > 0)) || ((positions[i - 1][j] > 0) && (positions[i][j + 1] > 0)) || ((positions[i - 1][j] > 0) && (positions[i][j - 1] > 0)) || ((positions[i][j - 1] > 0) && (positions[i][j + 1] > 0)))

    {

        positions[i][j] = 10;

        lengchange = algo(positions, endpathhoriz[0], endpathvert[0], endpathhoriz[1], endpathvert[1]);

        positions[i][j] = 0;

        if ((leng != -1) && (lengchange != -1))

        {

            if ((lengchange < leng) && (lengchange < bestleng))

            {

                bestleng = lengchange;

                agent = x;

                agentnewpositvert = i;

                agentnewposithoriz = j;

            };

        }

        else
```

```
                                        {
                                            if (lengchange != -1)
                                                if (lengchange < bestleng)
                                                    {
                                                        bestleng = lengchange;
                                                        agent = x;
                                                        agentnewpositvert = i;
                                                        agentnewposithoriz = j;
                                                    };
                                        };
                                };
                        };
                movingagents[agentpositvert[x]][agentposithoriz[x]] = 10;
                positions[agentpositvert[x]][agentposithoriz[x]] = 10;
        };//Searching best place for agents
        if (check == true)
                if (agent == -1)
                        return 0;
                else return -1;
```

```
if (bestleng < 100)
{
    switch (agent)
    {
        case 0:
        {
            cout << "Move agent from (" << agentpositvert[0] << "," << agentposithoriz[0] << ") to (" << agentnewpositvert << "," << agentnewposithoriz << ")" << endl;
            movingagents[agentpositvert[0]][agentposithoriz[0]] = 0;
            movingagents[agentnewpositvert][agentnewposithoriz] = 10;
            break;
        };
        case 1:
        {
            cout << "Move agent from (" << agentpositvert[1] << "," << agentposithoriz[1] << ") to (" << agentnewpositvert << "," << agentnewposithoriz << ")" << endl;
            movingagents[agentpositvert[1]][agentposithoriz[1]] = 0;
            movingagents[agentnewpositvert][agentnewposithoriz] = 10;
            break;
```

```
        };
};
switch (numberplayer)
{
case 1:
{
        for (int i = 0; i <= 10; i++)
                for (int j = 0; j <= 10; j++)
                        movingagentsfirst[i][j] =  movingagents[i][j];
        break;
};
case 2:
{
        for (int i = 0; i <= 10; i++)
                for (int j = 0; j <= 10; j++)
                        movingagentssecond[i][j] = movingagents[i][j];
        break;
};
case 3:
{
        for (int i = 0; i <= 10; i++)
```

```
                    for (int j = 0; j <= 10; j++)
                        movingagentsthird[i][j] = movingagents[i][j];
                break;
            };
            };
            cout << "New length of way for " << numberplayer <<" player is " << bestleng << endl;
            return bestleng;
        }
        else if (leng == -1) { cout << "Path is destroyed and not restored" << endl; return -1; }
        else { cout << "Nothing changes" << endl; return -1; }
};

int checkpath(float posit[11][11])
{
        vector<int> endpathhorizch, endpathvertch;
        int leng;
        for (int i = 0;i<=10;i++)
            for (int j = 0; j <= 10; j++)
            {
                if (posit[i][j] != 1)
```

```
                {
                        posit[i][j] = posit[i][j] + movingagentsfirst[i][j] + movingagentssecond[i][j] + movingagentsthird[i][j];
                }
                else
                {
                        endpathvertch.push_back(i);
                        endpathhorizch.push_back(j);
                };
        };
        leng = algo(posit, endpathhorizch[0], endpathvertch[0], endpathhorizch[1], endpathvertch[1]);
        return leng;
};

int main() {
        for (int i = 1; i <= 10; i++)
                for (int j = 1; j <= 10; j++)
                {
                        movingagentsfirst[i][j] = 0;
                        movingagentssecond[i][j] = 0;
                        movingagentsthird[i][j] = 0;
```

```
        };
    movingagentsfirst[0][1] = 10;
    movingagentsfirst[1][0] = 10;
    movingagentssecond[0][1] = 10;
    movingagentssecond[1][0] = 10;
    movingagentsthird[0][1] = 10;
    movingagentsthird[1][0] = 10;
    ifstream firstplayer("first.txt"), secondplayer("second.txt"), thirdplayer("third.txt");
    float firstplayerposit[11][11], secondplayerposit[11][11], thirdplayerposit[11][11];
    bool firstcheck[11][11], secondcheck[11][11], thirdcheck[11][11];
    int firstcheckcount[11][11], secondcheckcount[11][11], thirdcheckcount[11][11];
    int reliable=4;
    cout << "Begin positions:" << endl << "First:" << endl;
    if (firstplayer.is_open())
        for (int i = 1; i <= 10; i++)
        {
            for (int j = 1; j <= 10; j++)
            {
                firstplayer >> firstplayerposit[i][j];
```

```
                    firstcheck[i][j] = true;
                    cout << firstplayerposit[i][j] << " ";
                    firstcheckcount[i][j] = 0;
                };
                cout << endl;
        };
cout <<"Second:" <<endl;
if (secondplayer.is_open())
                for (int i = 1; i <= 10; i++)
                {
                    for (int j = 1; j <= 10; j++)
                    {
                        secondplayer >> secondplayerposit[i][j];
                        cout << secondplayerposit[i][j] << " ";
                        secondcheck[i][j] = true;
                        secondcheckcount[i][j] = 0;
                    };
                cout << endl;
                };
cout <<"Third:" <<endl;
        if (thirdplayer.is_open())
```

```
                for (int i = 1; i <= 10; i++)
                {
                        for (int j = 1; j <= 10; j++)
                        {
                                thirdplayer >> thirdplayerposit[i][j];
                                cout << thirdplayerposit[i][j] << " ";
                                thirdcheck[i][j] = true;
                                thirdcheckcount[i][j] = 0;
                        };
                        cout << endl;
                };
        cout << endl;
    firstplayer.close();
    secondplayer.close();
    thirdplayer.close();
    int countmove = 0;
    int a = checkpath(firstplayerposit);
    int b = checkpath(secondplayerposit);
    int c = checkpath(thirdplayerposit);
    cout <<"Begin length first: " <<a << ", second: " << b << ", third: " << c <<
endl;
    int first = moveagent(firstplayerposit, 1, true);
```

```
int second = moveagent(secondplayerposit, 2, true);
int third = moveagent(thirdplayerposit, 3, true);
float firstplayeravailable[11][11], secondplayeravailable[11][11], thirdplayeravailable[11][11];
while ((first == -1) || (second == -1) || (third == -1))
{
    countmove++;
    cout << countmove << " step" << endl;
    float randvalue;
    cout << "First player:";
    for (int i = 1; i <= 10; i++)
        for (int j = 1; j <= 10; j++)
        {
            randvalue = (rand() % 101) / 100;
            if (firstcheckcount[i][j] == reliable)
            {
                cout << " back (" << i << "," << j << ") ";
                firstcheck[i][j] = true;
                firstcheckcount[i][j] = 0;
            }
            else if (firstcheckcount[i][j] != 0) firstcheckcount[i][j]++;
            if (firstplayerposit[i][j] != 0)
```

```
                {
                        if (randvalue > firstplayerposit[i][j])
                        {
                                if (firstcheck[i][j] == true)
                                {
                                        cout << " out (" << i << "," << j << ")
";
                                        firstcheck[i][j] = false;
                                        firstcheckcount[i][j] = 1;
                                        firstplayeravailable[i][j] = 0;
                                };
                        }
                        else
                                if (firstcheck[i][j] == true)
                                        firstplayeravailable[i][j]                =
firstplayerposit[i][j];
                }
                        else firstplayeravailable[i][j] = 0;
                };
        cout << endl;
        first = moveagent(firstplayeravailable, 1, false);
        cout << "Second player:";
```

```
for (int i = 1; i <= 10; i++)
    for (int j = 1; j <= 10; j++)
    {
        randvalue = (rand() % 101) / 100;
        if (secondcheckcount[i][j] == reliable)
        {
            cout << " back (" << i << "," << j << ") ";
            secondcheck[i][j] = true;
            secondcheckcount[i][j] = 0;
        }
        else if (secondcheckcount[i][j] != 0) secondcheckcount[i][j]++;
        if (secondplayerposit[i][j] != 0)
        {
            if (randvalue > secondplayerposit[i][j])
            {
                if (secondcheck[i][j] == true)
                {
                    cout << " out (" << i << "," << j << ") ";
                    secondcheck[i][j] = false;
                    secondcheckcount[i][j] = 1;
```

```
                    secondplayeravailable[i][j] = 0;
                };
            }
            else
                if (secondcheck[i][j] == true)
                    secondplayeravailable[i][j] = secondplayerposit[i][j];
            }
            else secondplayeravailable[i][j] = 0;
        };
    cout << endl;
    second = moveagent(secondplayeravailable, 2, false);
    cout << "Third player:";
    for (int i = 1; i <= 10; i++)
        for (int j = 1; j <= 10; j++)
        {
            randvalue = (rand() % 101) / 100;
            if (thirdcheckcount[i][j] == reliable)
            {
                cout << " back (" << i << "," << j << ") ";
                thirdcheck[i][j] = true;
                thirdcheckcount[i][j] = 0;
```

```
                    }
                    else if (thirdcheckcount[i][j] != 0)
thirdcheckcount[i][j]++;
                    if (thirdplayerposit[i][j] != 0)
                    {
                        if (randvalue > thirdplayerposit[i][j])
                        {
                            if (thirdcheck[i][j] == true)
                            {
                                cout << " out (" << i << "," << j << ")
";
                                thirdcheck[i][j] = false;
                                thirdcheckcount[i][j] = 1;
                                thirdplayeravailable[i][j] = 0;
                            };
                        }
                        else
                            if (thirdcheck[i][j] == true)
                                thirdplayeravailable[i][j] =
thirdplayerposit[i][j];
                    }
                    else thirdplayeravailable[i][j] = 0;
```

```
            };
        cout << endl;
        third = moveagent(thirdplayeravailable, 3, false);
        first = moveagent(firstplayeravailable, 1, true);
        second = moveagent(secondplayeravailable, 2, true);
        third = moveagent(thirdplayeravailable, 3, true);
};//Поиск неулучшаемых позиции
for (int i = 1; i <= 10; i++)
        for (int j = 1; j <= 10; j++)
        {
                if (firstcheck[i][j] == true)
                {
                        firstplayeravailable[i][j] = firstplayerposit[i][j];
                }
                else firstplayeravailable[i][j] = 0;
                if (secondcheck[i][j] == true)
                {
                        secondplayeravailable[i][j] = secondplayerposit[i][j];
                }
                else secondplayeravailable[i][j] = 0;
                if (thirdcheck[i][j] == true)
```

```
                {
                        thirdplayeravailable[i][j] = thirdplayerposit[i][j];
                }
                else thirdplayeravailable[i][j] = 0;
        };
        cout << "End positions:" << endl << "First:" << endl;
        for (int i = 1; i <= 10; i++)
        {
                for (int j = 1; j <= 10; j++)
                {
                        if (((movingagentsfirst[i][j] != 0) || (movingagentssecond[i][j] != 0) || (movingagentsthird[i][j] != 0)) && (firstplayeravailable[i][j] != 1)) firstplayeravailable[i][j] = 10;
                        cout << firstplayeravailable[i][j] << " ";
                };
                cout << endl;
        };
        cout << "Second:" << endl;
        for (int i = 1; i <= 10; i++)
        {
                for (int j = 1; j <= 10; j++)
                {
```

```
                if (((movingagentsfirst[i][j] != 0) || (movingagentssecond[i][j] != 0) || (movingagentsthird[i][j] != 0)) && (secondplayeravailable[i][j] != 1)) secondplayeravailable[i][j] = 10;

                cout << secondplayeravailable[i][j] << " ";

            };

            cout << endl;

        };

        cout << "Third:" << endl;

        for (int i = 1; i <= 10; i++)

        {

            for (int j = 1; j <= 10; j++)

            {

                if (((movingagentsfirst[i][j] != 0) || (movingagentssecond[i][j] != 0) || (movingagentsthird[i][j] != 0)) && (thirdplayeravailable[i][j] != 1)) thirdplayeravailable[i][j] = 10;

                cout << thirdplayeravailable[i][j] << " ";

            };

            cout << endl;

        };

        cout << endl;

        a = checkpath(firstplayeravailable);

        b = checkpath(secondplayeravailable);

        c = checkpath(thirdplayeravailable);
```

```
        if ((a == -1) || (b == -1) || (c == -1))
        {
                cout << "Exception " << endl;
        }
        else cout << "Nash equilibrium - Exist"<< endl;
        system("pause");
        return 0;
}
```